Amendments to the Specification:

Please replace paragraph [0066], [0067], [0068], and [0069] with the following amended paragraph:

[0066] **Figure 11B** shows in further detail the net kernel software being executed by each processor, such as processor 102A of **Figure 6A**. The net kernel software 431 includes a scheduler module 445 as well as the processor's socket ISM module 449 for communication with the host 439. The net kernel software 431 also includes a messaging queue manager 447 which manages the interprocessor messages 441 as part of the interprocessor communication through bus 103 as shown in **Figure 11A**. The software 431 further includes, in this embodiment, TCP processing software as well as IP processing ~~software 451 and 543~~ software 451 and 453, respectively. The network protocol processing software is in communication with, in this case, a network communication port which is an Ethernet port 437. The software 431 also maintains interrupt status registers 443 which receive interrupts from the interrupt controller 433 which concentrates interrupts 435 from various interrupt sources including the timeout timers, socket calls from the host and outgoing packets from the host, interrupts from other processors for interprocessor communication, and interrupts generated by the dispatch logic (e.g. dispatch logic 106 in the case of packet reception from the network). The operating system portion of the net kernel software 431 provides basic functions such as scheduling and dispatching, memory management, timing services, thread management, synchronization, and system initialization. It supports execution at two levels, interrupts and threads, and the execution priorities are such that interrupts, unless disabled, are processed with the highest priority. High priority threads have the second highest priority and such threads are not destroyed. Instead they are blocked on return and reentered when subsequently resumed. Normal or low priority threads may be preempted by either the interrupts or the higher priority threads. Normal or low priority threads are terminated when they complete and thus, if necessary, should be written as a function that runs forever. In one embodiment, the net kernel performs all network protocol processing at the interrupt level in order to minimize context switching. The net kernel also polls the Ethernet MAC interface (or other network communication medium interface) as well as the DMA interface to the processor's memory (e.g. memory 111 of **Figure 6A**) or to the host's memory.

[0067] **Figure 12A** shows an example of a method of load balancing of the packet flows through 4 processors of a network protocol processing system such as the system 101. The host OSM module 503 controls the distribution of outgoing packets 501 to one of the four processors, each of which are shown in **Figure 12A** as having separate sockets (as in the case of **Figure 9**); the sockets of **Figure 12A** are shown as 505A, 505B, 505C, and 505N. The host OSM module remembers for a session the particular processor which processed the incoming packets which are being responded to by the outgoing packets 501. The host OSM will recall for this session the identification of the particular processor and forward the outgoing packets to that processor. The host OSM typically employs a tag for each session which identifies the particular processor in the network protocol processing system. If there is no tag (e.g. in the case where the server initiates a session) then the host OSM will provide a tag to identify the particular processor in the network protocol processing system to process the outgoing packet. However, processing at the IP layer (blocks 511A, 511B, 511C, or 511N) may be performed by any one of the available processors in the network protocol processing system. Furthermore, datagrams originating from connectionless protocols such as UDP and IGMP can be processed by any one of the executing network protocol stacks. Incoming packets 515 are hashed in a hashing operation 513 which may be performed by simple programmable hardware logic that dispatches received IP packets to one of the 4 processors for processing of the IP protocol in blocks 511A, 511B, 511C, or 511N. After the IP layer processing, the transport protocol header of the IP packet is examined. If the datagram belongs to connection oriented protocol such as a TCP protocol, a software hash function may route the datagram to the correct processor 507 for transport protocol processing. Thus all datagrams belonging to a TCP session are restricted to be handled by the same processor where the session was initiated or opened. On the other hand, if the datagram is not specific to a connection (e.g. a connectionless protocol) then the datagram may be rerouted in order to balance the load by the load balancing module 509.

[0068] **Figure 12B** shows a further example of the direction of packets based on address hashing and load balancing. The hash function 533 examines the source IP and source port number 531 and determines, in the case of non-fragmented packets, the proper processor which is selected to process the packet pursuant to a network protocol. In the case shown in **Figure 12B**, the processor selected by the hash function 533 is the processor P0, but as shown in **Figure 12B**, the processor P0 determines in operation 535 that its processing queue is too large and forwards an interprocessor communication request to another processor, in this case P2 539, to process the packet. Message queues are maintained between the processors, such as message queue 537 between processor P0 and processor P2 and message queue 547 between processor P1 545 and processor P0. The hash function 533 is typically an exclusive OR which produces two bits to select a processor in the case where the network processing system includes 4 processors such as processors 102A, 102B, 102C and 102N as shown in **Figure 7**. The hash function 543 is a different hash function which is employed for fragmented packets. In this case the IP identifier which identifies the session causes, through the hash function 543, the packet fragment to be directed to the same processor as prior fragmented packets having the same IP identifier. It can be seen from **Figure 12B** that fragmented packets 541 are processed through the hash function 543 and forwarded to processor P1 for processing. The processor P1 may indicate through its software message queue 547 that it is busy, which causes the processor P0 to forward packets it receives to processor P2 instead of processor P1.

[0069] A description of the OSM and ISM modules in the I2O architecture will now be provided while referring to **Figure 13**. In the following discussion, it is assumed that the I2O architecture with OSM and ISM modules are used to control the communication between the network protocol processing system, such as system 101, and a host processor <u>561;</u> it will be appreciated, however, that alternative architectures may be employed. Communication between a network protocol processing system such as system 101 which includes multiple processors, each executing a separate network protocol stack as a separate thread, and a host processor or processors is based on the I2O socket architecture which defines a messaging framework for two systems to exchange information with each other. This architecture is well known and was developed by the I2O special interest group. Detailed information regarding this architecture can be found at the I2O web site which is www.intelligent-IO.com. To support the I2O socket architecture, a socket ISM module is running at the top of the protocol stack on each processor. The ISM module interacts with a host OSM module, intercepts all the socket calls, maintains and manages data structures for socket operations, and controls the data moving into and from the host. A socket layer in each ISM of each processor in a network protocol processing system provides an interface (API--application program interface) used by applications running on the host processor to access TCP/IP services. The OSM and ISM provide the communication between the socket API and the network protocol stack. TCP or UDP sockets are maintained and synchronized between the host and each processor. A set of message queues is managed by the ISM modules and the host OSM module as shown in **Figure 13**. For each processor in a network protocol processing system, there is a corresponding inbound free queue and an inbound post queue. In the case shown in **Figure 13**, there are 4 processors in the network protocol processing system (and thus the system of **Figure 13** resembles the embodiment shown in **Figure 7** which includes 4 processors 102A, 102B, 102C, and 102N). ISM module 567A is executing on processor P0 while ISM modules 567B, 567C, and 567D are executing respectively on processors P1, P2, and P3. The ISM's of the 4 processors jointly control, as represented by the block 569, the outbound free queue 571 and the outbound post queue 573 which allows for the transmission of messages to the host OSM from any one of the ISMs executing on one of the processors. Processor P0 has its corresponding inbound free queues and inbound post queues 563A and 565A, and each of the processors P1, P2, and P3 have their respective inbound free queues and inbound post queues (563B, 565B, 563C, 565C, 563N, and 565N). Each queue may contain a plurality of message frame addresses which are pointers to a memory address that contains a message that needs to be processed. In the case of the outbound free queue 571 and the outbound post queue 573, these message frame addresses are pointers to locations in the host's memory. In the case of the other queues shown in **Figure 13** (563A, 563B, 563C, 563N, 565A, 565B, 565C, and 565N), these message frame addresses are pointers to memory locations of the corresponding processor's memory such as preallocated memory portions of the memory 111 of **Figure 6A**. When the OSM host has a message to communicate to a particular processor, such as processor P0, the OSM host determines whether the processor's corresponding inbound free queue (e.g. queue 563A in the case of processor P0) has a free entry, and if so the inbound free queue provides a free MFA (message frame address) to the OSM host and the OSM host then causes a DMA operation to occur, typically from the host's memory (e.g. memory 121 of **Figure 6B**) to a preallocated portion of the network protocol processing system's memory, such as the memory 111. When the DMA operation is complete, the OSM host posts the MFA address in the corresponding inbound post queue for that processor (e.g. queue 565A for processor P0) and this queue can then interrupt its corresponding processor or its processor can poll the queue to see if there are any messages to process. A similar sequence of operations occurs in the reverse direction when an ISM module on a particular processor seeks to send a message or communicate data to the host OSM. In this case, the particular ISM module asks the outbound free queue whether there are any available MFAs, and if so, the outbound free queue 571 provides an available MFA to the requesting ISM, which in turn causes a DMA operation to transmit data from the processor's memory (e.g. a preallocated portion of the memory 111) to the host's memory. After the DMA operation is complete, the particular ISM posts the MFA in the outbound post queue 573 which can cause an interrupt of the host processor or the host processor can poll the queue to see if there are messages to process. It will be appreciated that the information which is exchanged in this architecture can include either data or commands which can be interpreted upon receipt to cause a particular action or a combination of data and commands.